Benchmarking tools for SKAO Pipelines

Anass Serhani
Avalon, Inria
anass.serhani@inria.fr

Shan Mignot
Lagrange, OCA
shan.mignot@oca.eu

SCOOP
DP ART Team
SKA Software Engineering

Joint work with: C. Sakka, A. Doussout and M. Stutz

ECLAT - Atelier Technique

November 28th, 2024

Outline

- **SDP Benchmark Suite**
  - Overview
  - Benchmark-Monitor (benchmon)
  - HPCToolkit

- **SKAO Pipelines**
  - Low pipeline
  - Preliminary benchmarking results
    - Calibrate
    - Predict
    - Image
    - 9 iterations

Benchmarking Tools

SDP Benchmark Suite | Overview

https://gitlab.com/ska-telescope/sdp/ska-sdp-benchmark-tests

https://developer.skao.int/projects/ska-sdp-benchmark-tests/

The regression test pipeline.

Goal

- Efficient benchmarking of SKA Pipelines
- Support procurement decision for the SDP

Available Apps

- Kernel benchmarks (*level 0)*:
  FFT - HPL - HPCG - Stream
- Pipelines components *(level 1)*:
  IDG - imaging_iotest
- Entire pipelines/workflows *(level 2)*:
  Low Iterative Calibration Pipeline

Software stack

- *Package management:*
  Spack + Guix
- *Testing framework:*
  ReFrame
- *Python environments*
  CONDA PIP
- *Scheduler*
  slurm workload manager

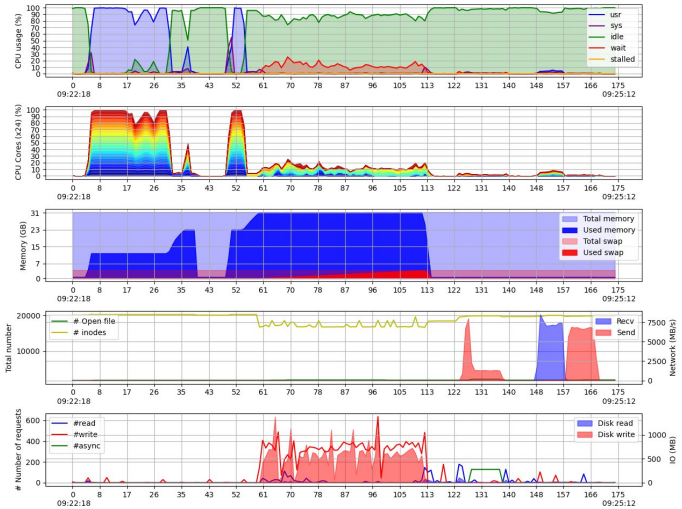
Partially tested on

Grid'5000 IRIS CSCS

Benchmark-Monitor | benchmon

Develop a **user-friendly tool** to:

- Detect bottlenecks and hotspots;
- Diagnose issues;
- Optimize performance.



Insights

Hardware + Software contexts

System monitoring

- CPU usage
- Memory/Swap space
- Disk reads/writes
- Network activity

Power consumption

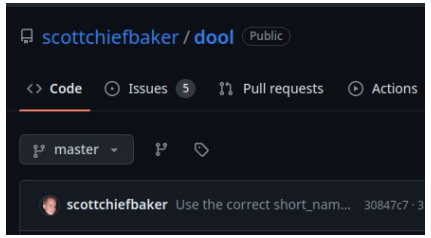
- Entire Socket
- Cores
- DRAM

Callstack record

- By level
- By thread

dool - System monitoring

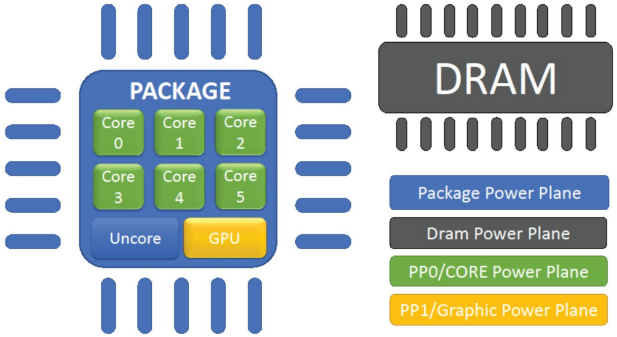
- CPU average/per-core usage
- Memory/Swap space usage
- Disk reads/writes, #inodes, #openfiles
- Network (send/recv, bandwith)




```
:dool --more 15
----total-cpu-usage---- -dsk/total- -net/total- ---------memory-usage--------- ---procs--- --load-avg--- ------system------
usr sys idl wai stl  read  writ  recv  send  used  free  cach  avai run blk new  1m   5m   15m        time
  0   0  99   0   0  14M 1421k    0     0  828M  183M 13.7G 14.1G   0   0 0.4 0.01 0.01    0 Sep-26 15:52:53
  0   0 100   0   0     0     0  69k   47k  828M  183M 13.7G 14.1G 2.0   0 0.1 0.01    0    0 Sep-26 15:53:08
  0   0 100   0   0     0  201k  53k   36k  828M  183M 13.7G 14.1G 3.0   0 0.1    0    0    0 Sep-26 15:53:23
  0   0 100   0   0     0     0  66k   45k  828M  183M 13.7G 14.1G 2.0   0 0.1    0    0    0 Sep-26 15:53:38
  0   0 100   0   0     0  100k  58k   41k  828M  183M 13.7G 14.1G   0   0 0.2    0    0    0 Sep-26 15:53:53
  0   0 100   0   0     0     0  64k   42k  828M  183M 13.7G 14.1G 1.0   0 0.3    0    0    0 Sep-26 15:54:00
```

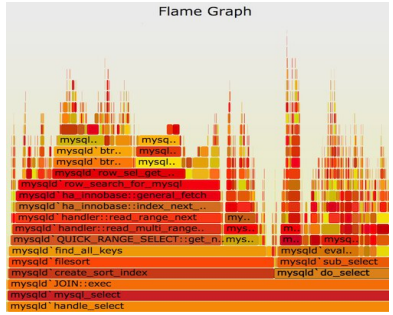
perf - Power Consumption

- With *perf* event
- power/energy-cores/
- power/energy-dram/
- power/energy-pkg/
- power/energy-gpu/



perf - Callstack recording

- *perf record;*
- *perf script*



Python Visualization + Flame Graph

Benchmon | Getting started

❖ Requirements

- ✓ *dool*
- ✓ *perf* + *perf* permissions
- ✓ *Python:* *matplotlib*, *numpy*, *yaml*, *csv*

❖ Installation

- Download

ska-telescope/sdp/ska-sdp-benchmark-monitor.git

- Executables

```
./bin/benchmon-software
./bin/benchmon-hardware
./bin/benchmon-start
./bin/benchmon-stop
./bin/benchmon-visu
```

❖ Example

```
# start benchmon
benchmon-start --system --power --call

# run App
./ft.A.x

# stop benchmon
benchmon-stop

# Create visualization figure
benchmon-visu --cpu --cpu-all --mem --call
```

❖ Start monitoring

benchmon-start

- [--traces-repo] Traces repository
- [--system] Record system resources usage
- [--system-sampling-interval] Recording delay (in seconds)
- [--power] Record power consumption
- [--power-delay-interval] Recording delay (in miliseconds)
- [--call] Record callstack
- [--call-mode] Recording mode
- [--call-profiling-frequency] Recording frequency (in Hz))
- [--sudo] sudo command for perf (if needed)

❖ Stop monitoring

benchmon-stop

❖ Visualizing

benchmon-visu

- [--traces-repo] Traces repository
- [--cpu] Draw CPU average usage (usr, sys, idle, wait)
- [--cpu-all] Draw CPU usage per core
- [--mem] Draw memory/swap usage
- [--net]Draw network activity (send/recv)
- [--io] Draw io activity (read/write)
- [--pow] Draw power profile
- [--call] Draw callstack
- [--call-depth|--call-depths] Set callstack depth
- [--fig-fmt] Figure format
- [--dpi] Figure dpi
- [--fig-path] Figure path

Benchmon | Example on FFT kernel

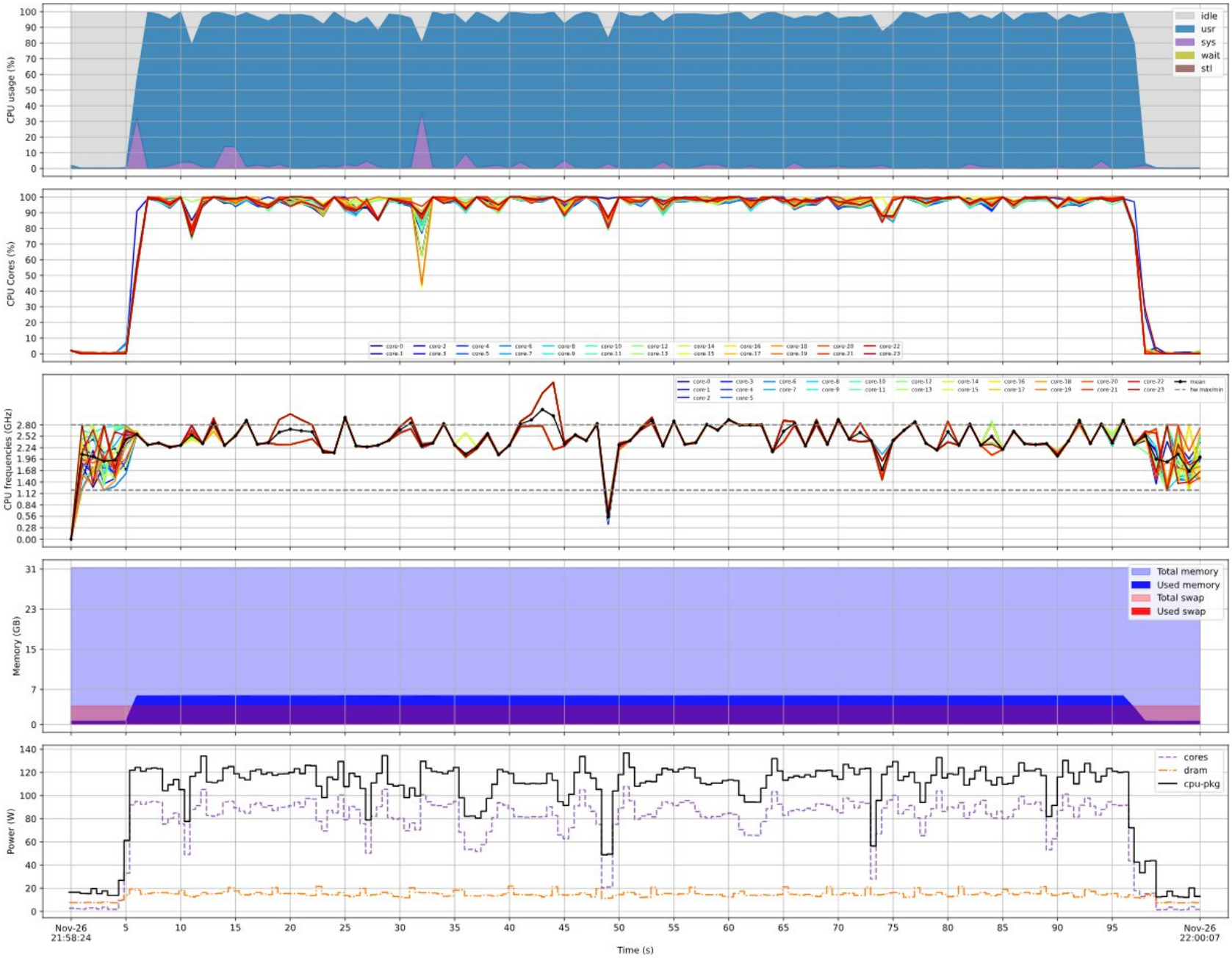
CPU usage

CPU usage per core

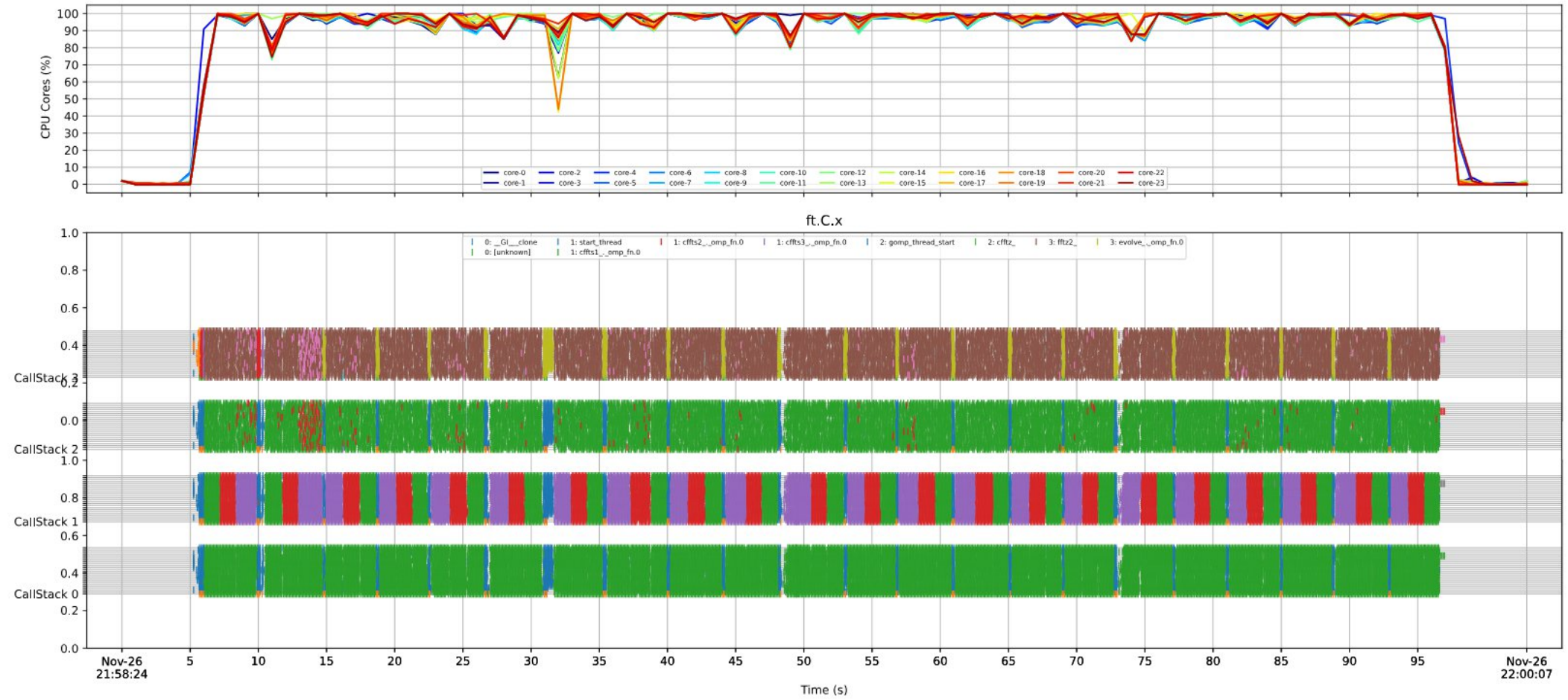
CPU frequencies per core

Memory & Swap

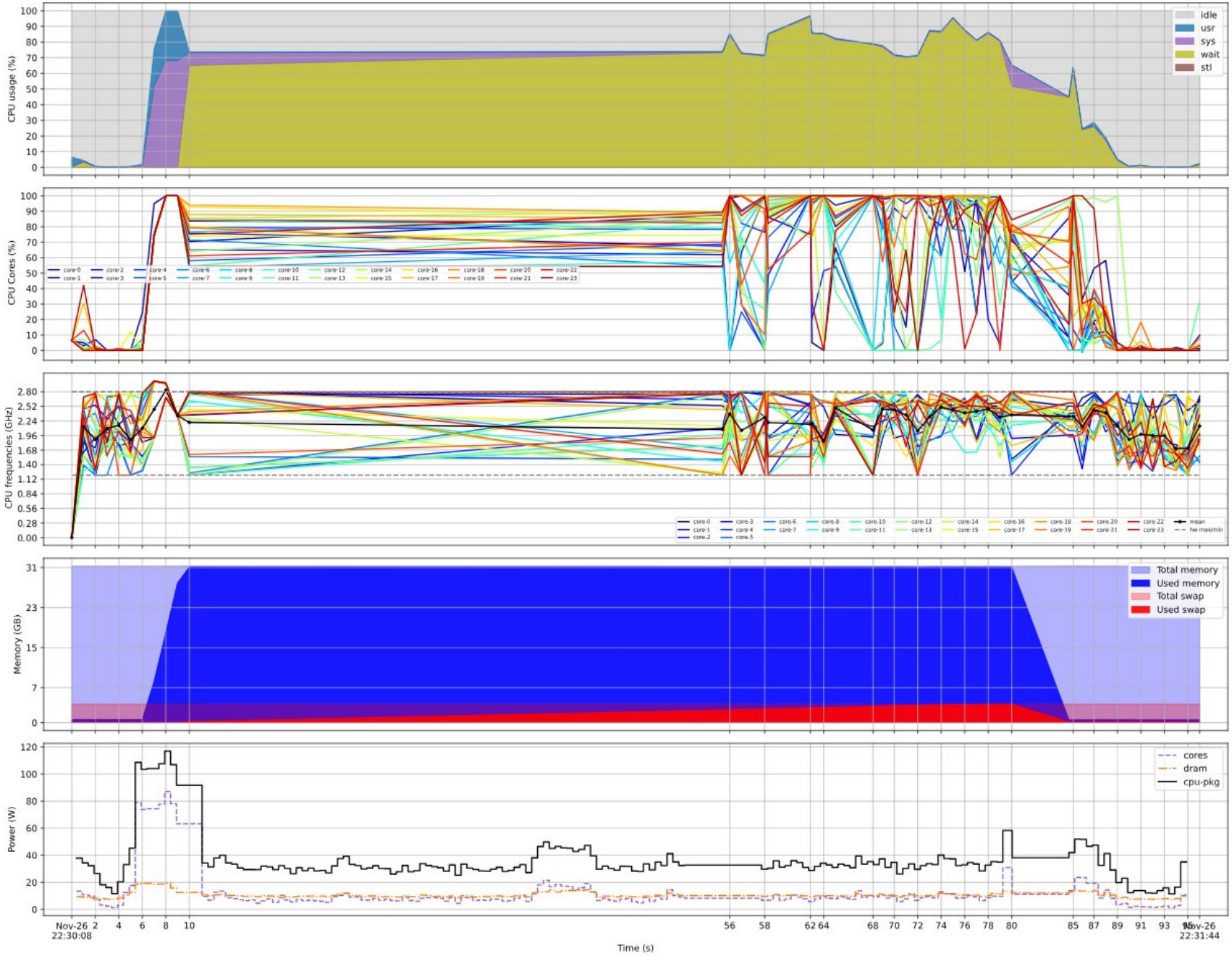
Energy



Benchmon | Example on FFT kernel



Benchmon | Example on FFT kernel



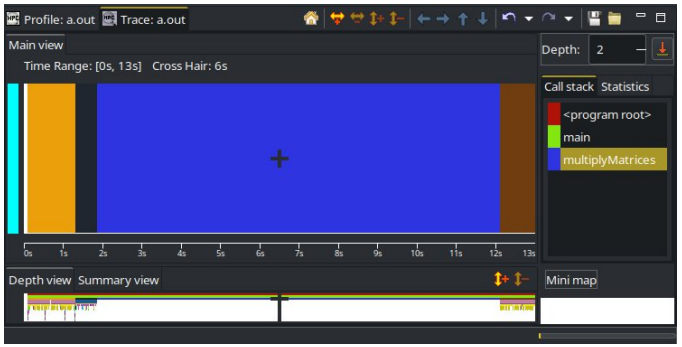
Benchmon | Example on FFT kernel

HPCToolkit

HPCToolkit

http://hpctoolkit.org/
https://gitlab.com/hpctoolkit/hpctoolkit

HPCToolkit is a suite of tools designed for performance analysis of applications running on parallel computers, offering detailed insights into resource usage and optimization opportunities.

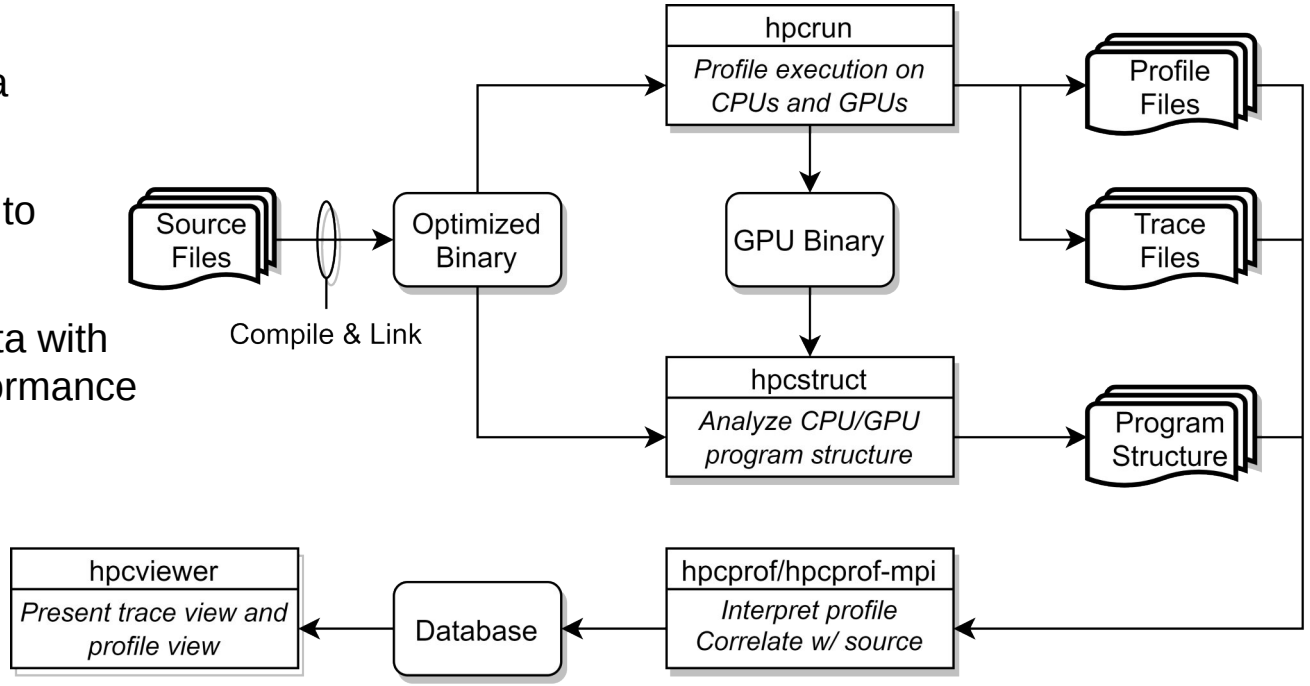


hpcrun: collect performance data from a program while it runs.

hpcstruct: analyzes a program's binary to understand its structure

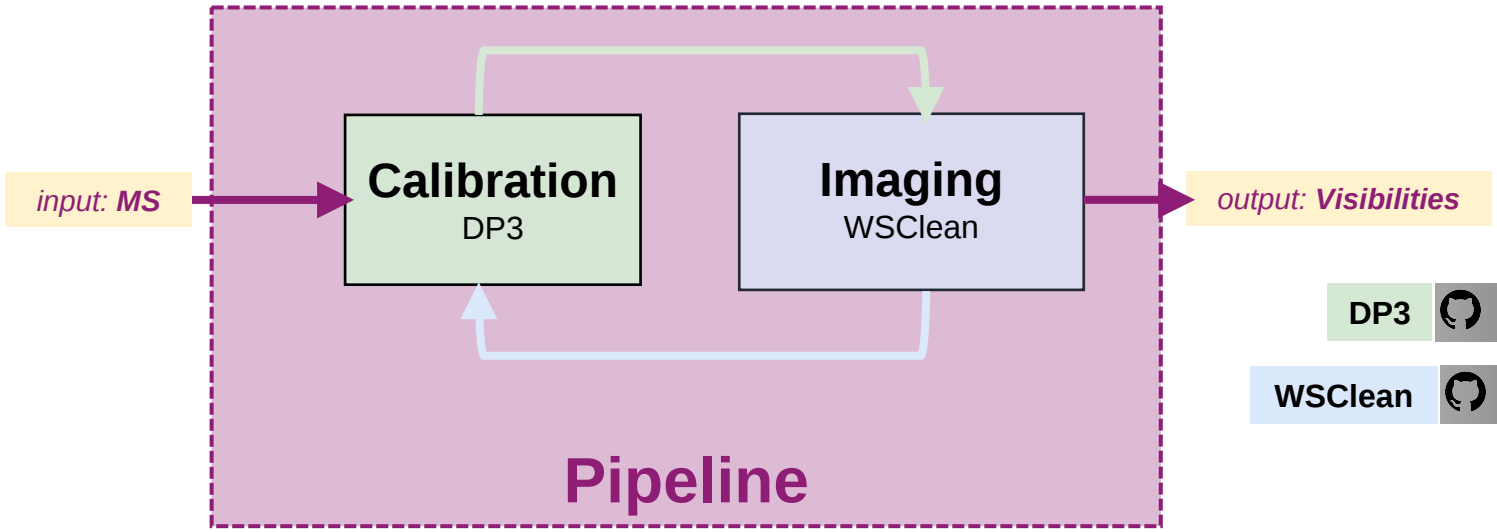
hpcprof: correlates the performance data with the program structure, producing a performance database.

hpcviewer: graphical interface to read the performance databases



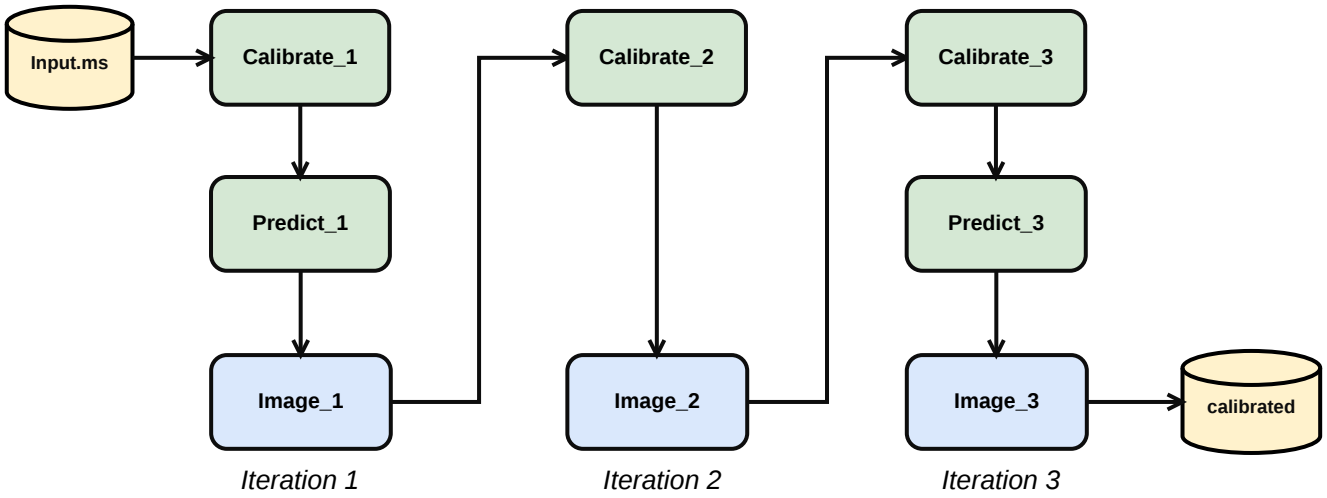
SKAO Pipelines

ska-telescope/sdp/science-pipeline-workflows/ska-sdp-wflow-low-selfcal



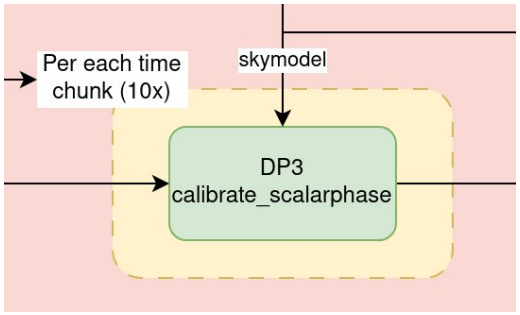
DP3 https://github.com/lofar-astron/DP3

WSClean https://github.com/aroffringa/wsclean

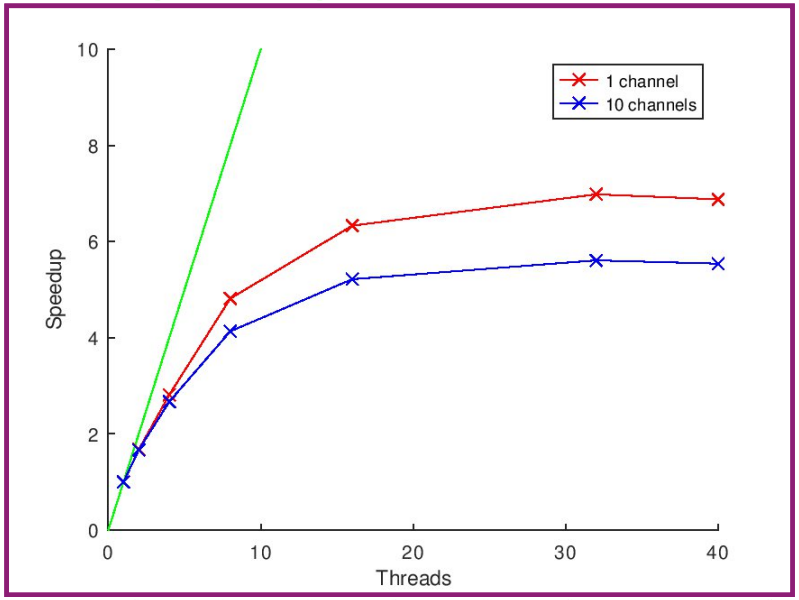


Low ICal | Calibrate 1: scaling

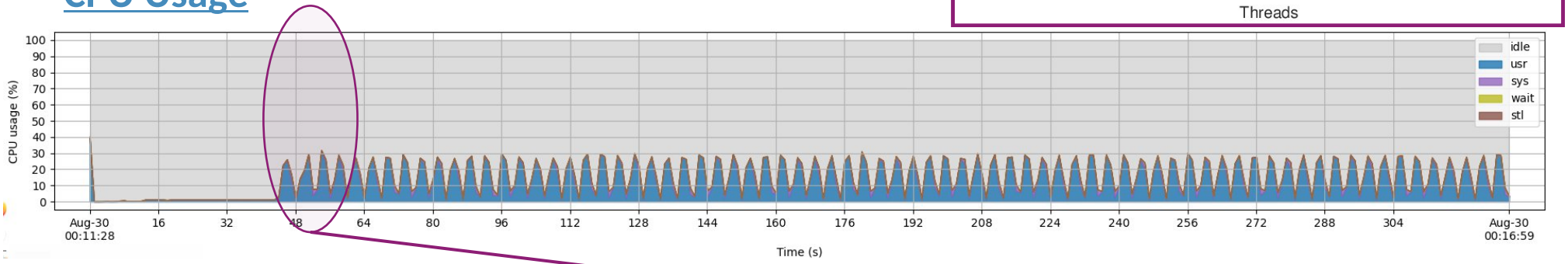
- **DP3 version 6.1.0**
- **Cascade Lake Node:**
  - Intel Cascade Lake 6248 CPU @ 2.50 GHz
  - Dual-Socket CPU, 20 cores per socket
  - 192 GB Memory
  - no hyperthreading
  - exclusive access

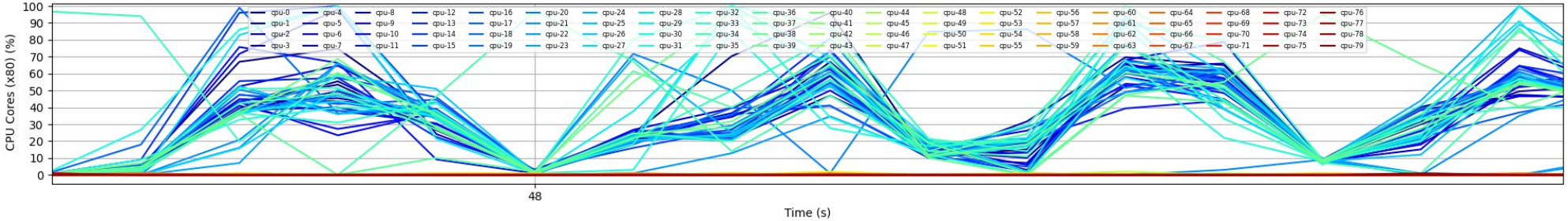


Strong scaling

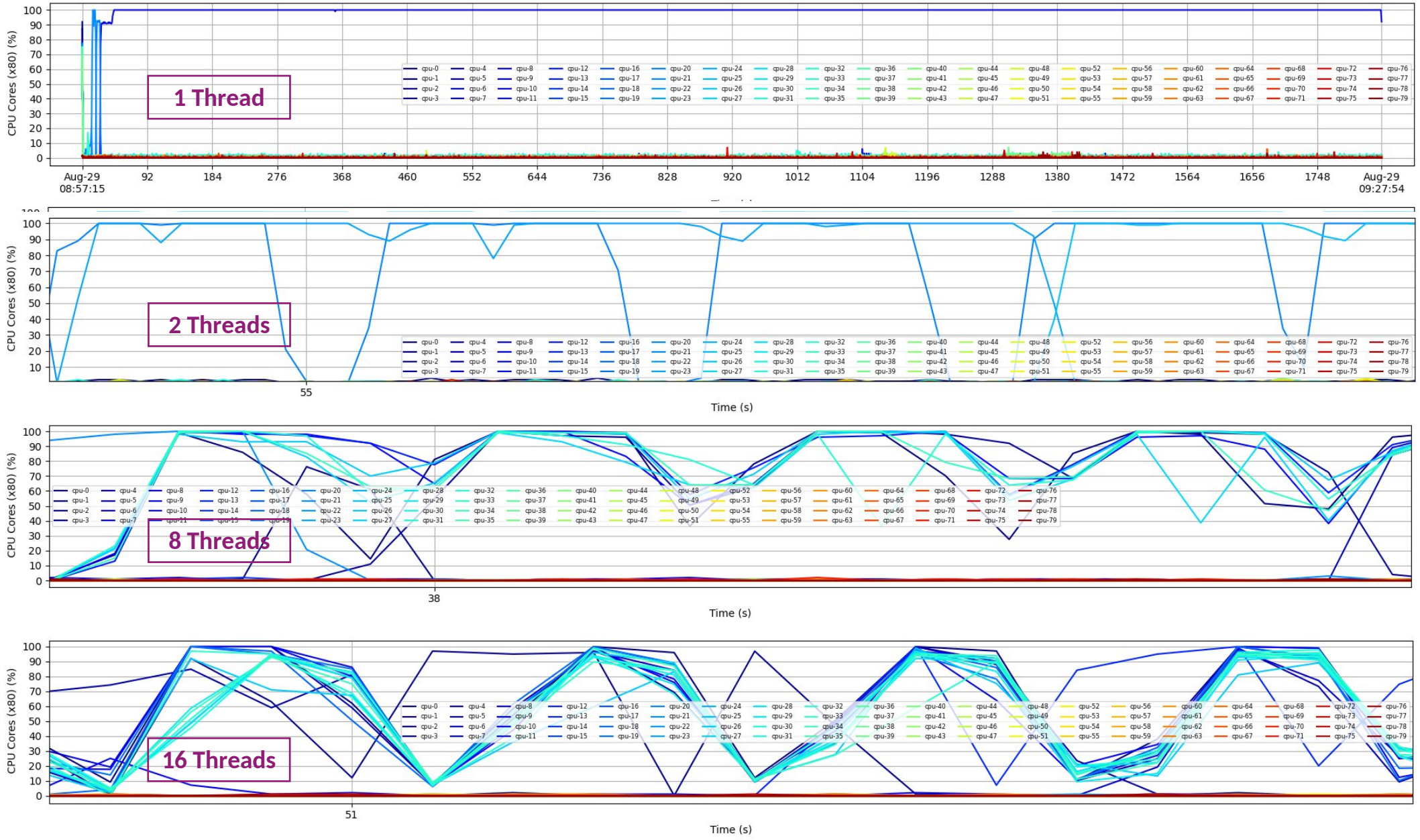


CPU Usage

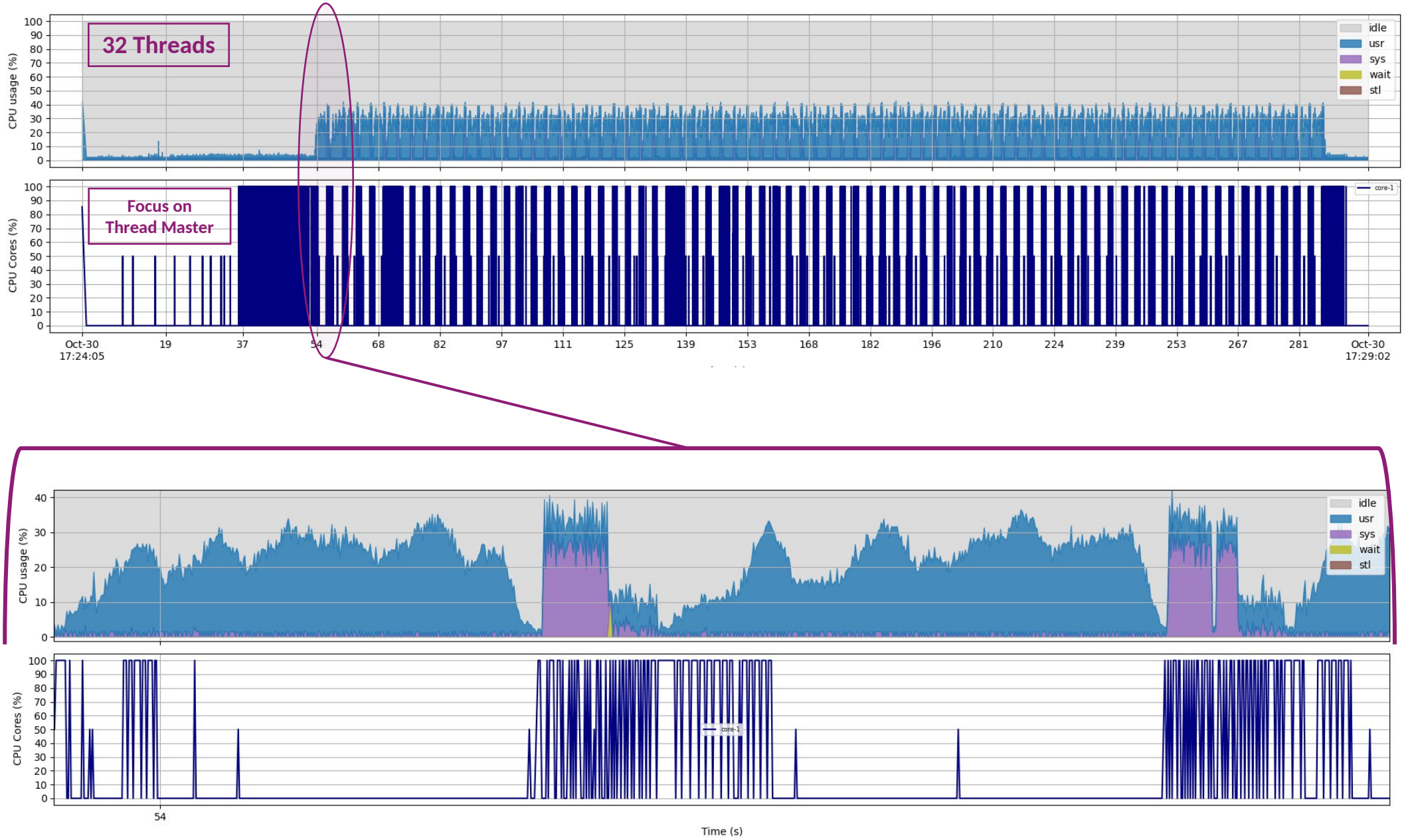




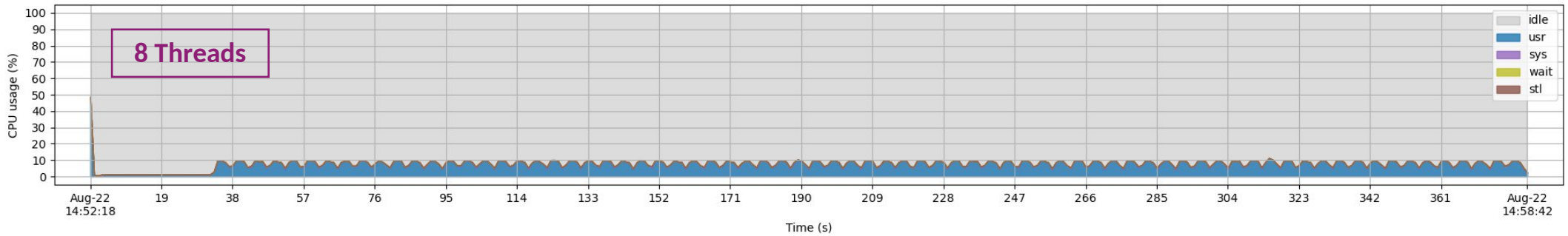
Low ICal | wsclean: threads activity

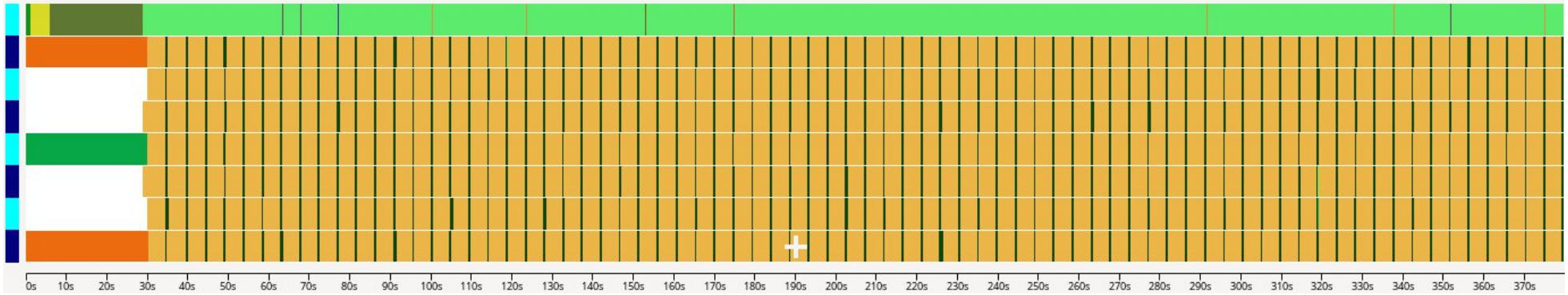


Low ICal | Calibrate 1: High Freq Mon

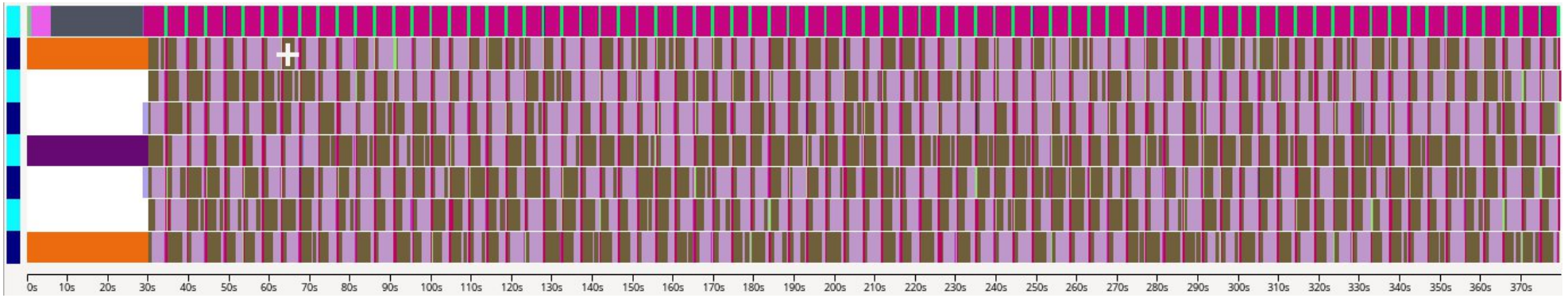


Low lCal | Calibrate 1: Tracing

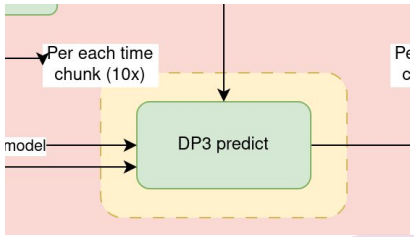




Chronogram of calls at **depth 4** for 8 requested threads and 1 channel: **aocommon::RecursiveFor** and **aocommon:StaticFor**

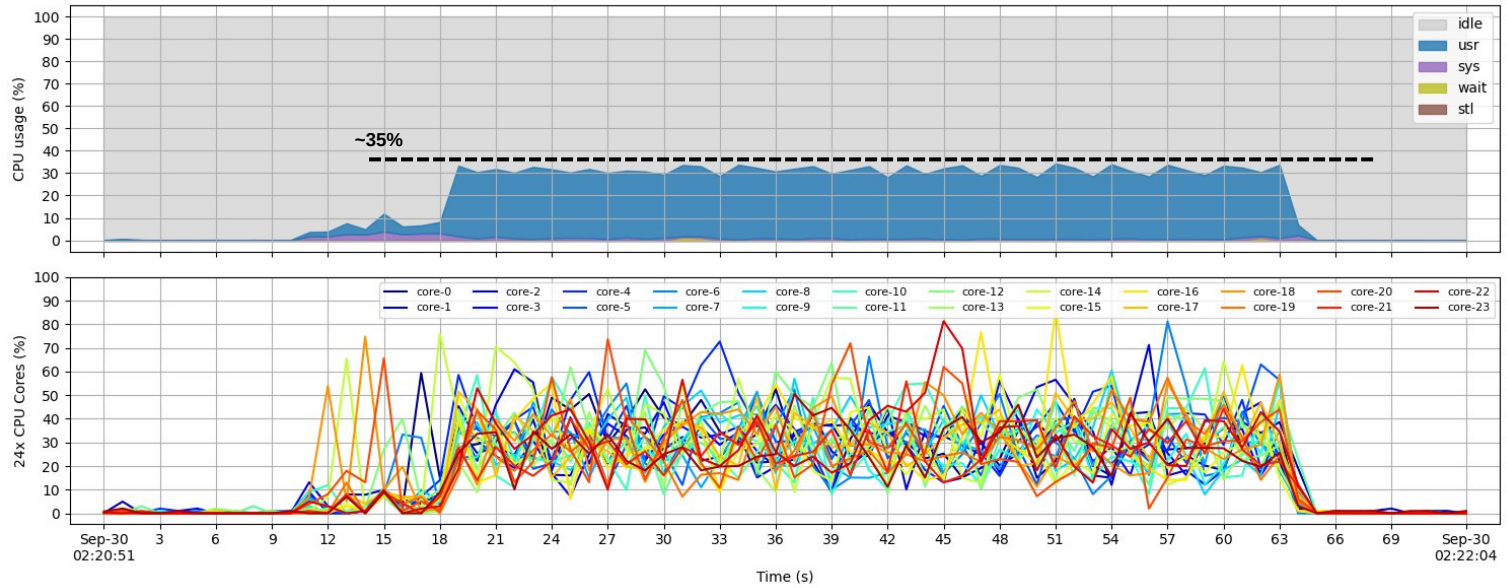


Chronogram of calls at **depth 5** for 8 requested threads and 1 channel. **dp3::steps:DDECal::doPrepare::_M_invoke** and **aocommon:StaticFor**

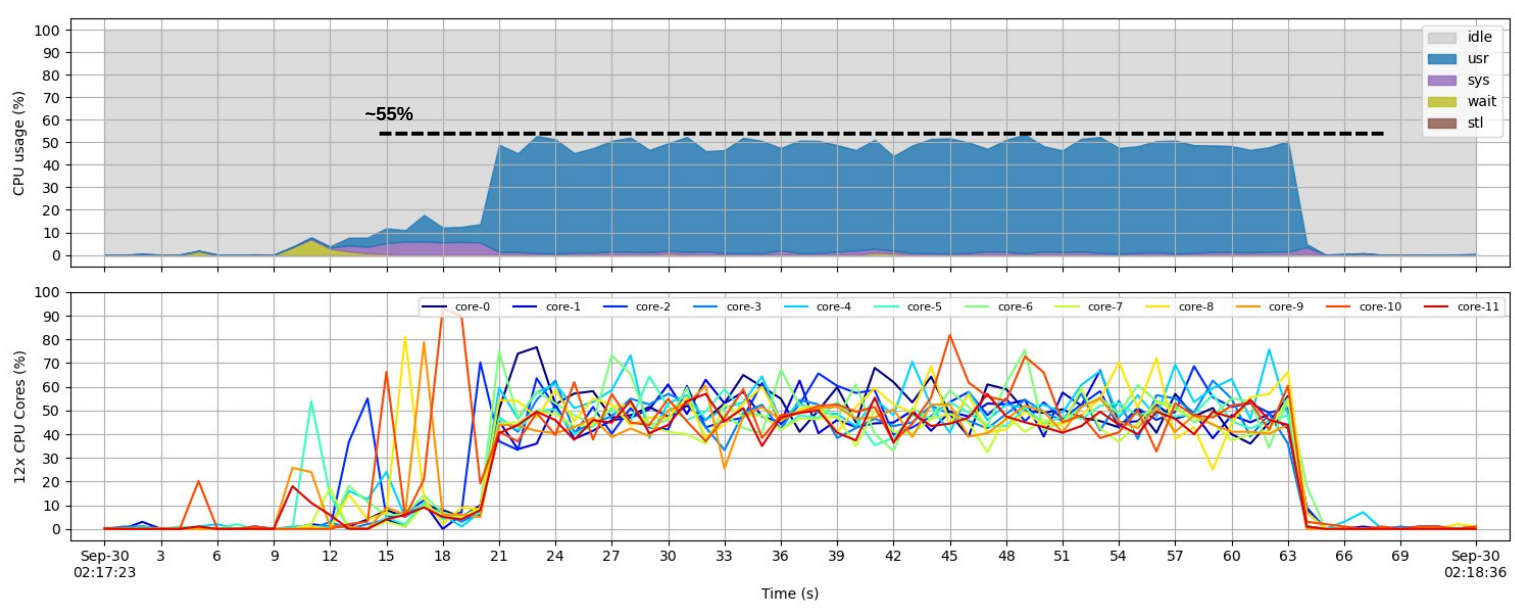


Predict 1: 9x Loop over time chunks

Hyper-threading
1x thread ~ 1x logical core
2x threads ~ 1x physical core

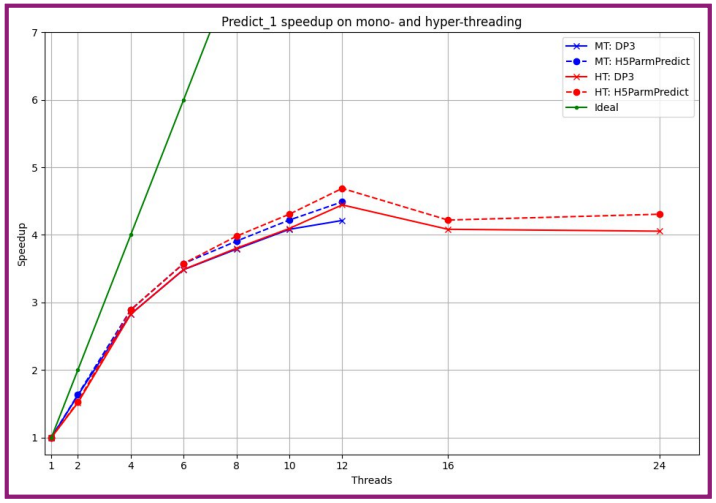


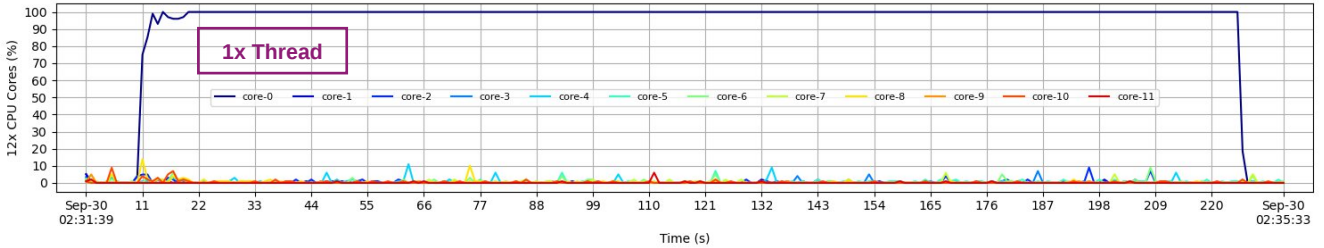
Mono-threading
1thread ~ 1 physical core

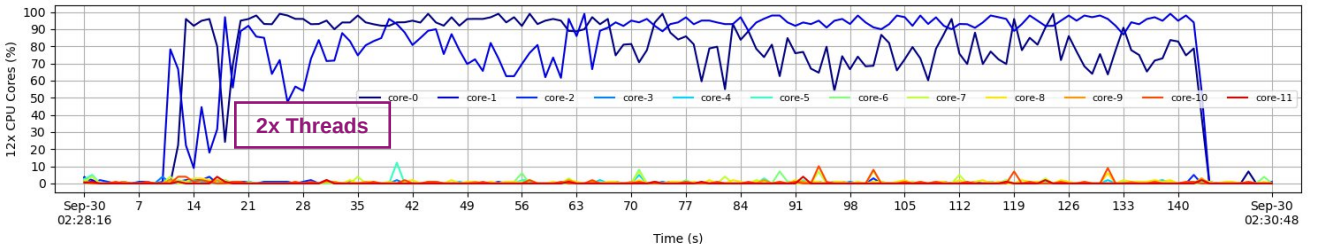


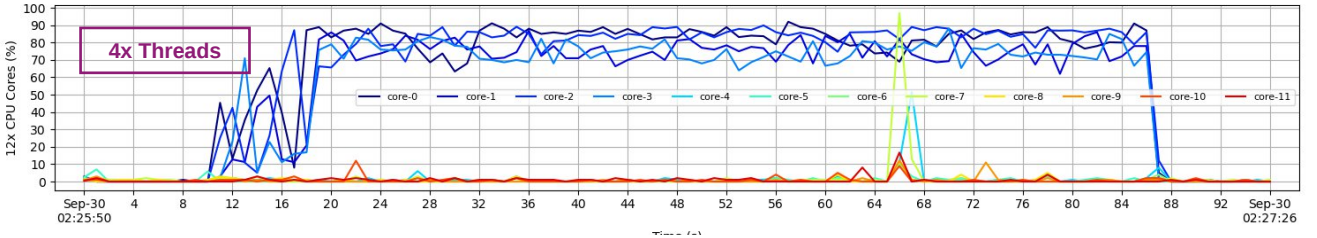
Low ICal | Predict 1: threads activity

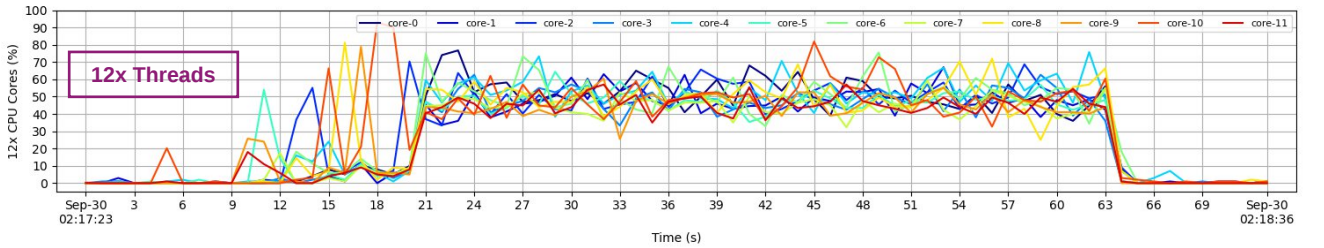
Strong Scaling

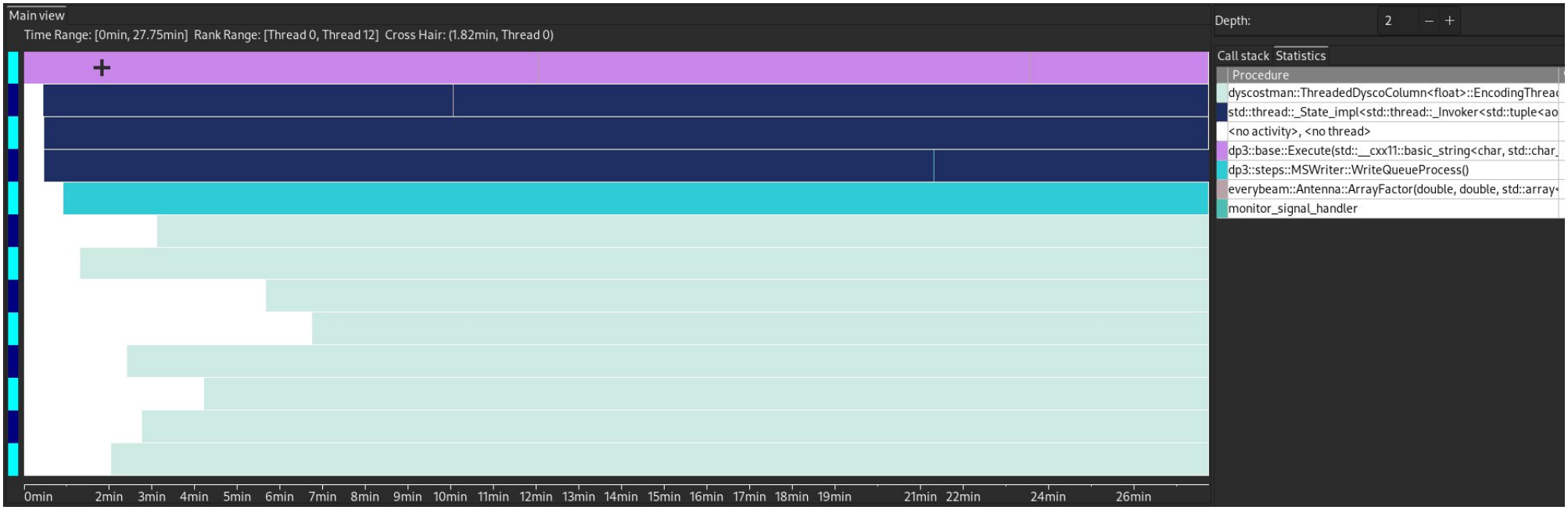






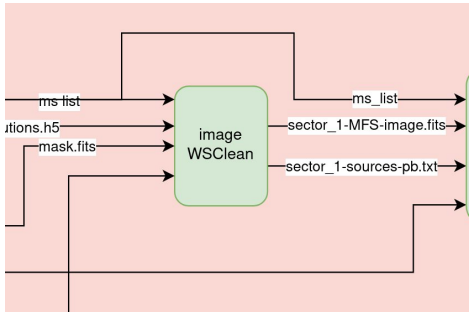






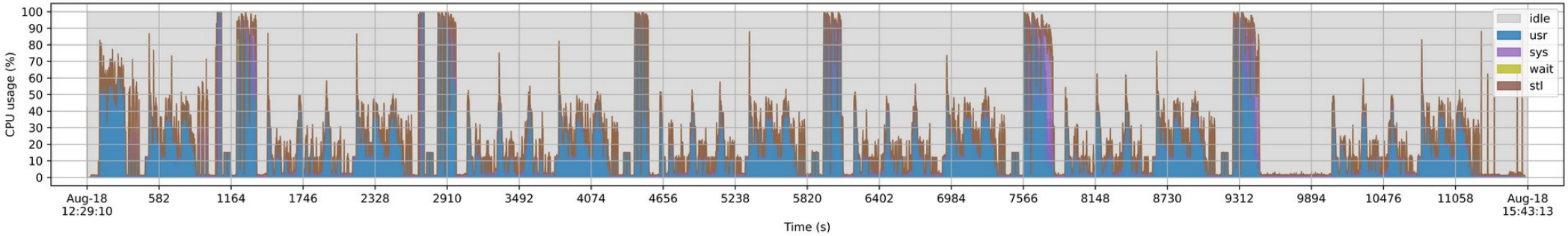
predict_1 created 13 threads instead of 4 threads.

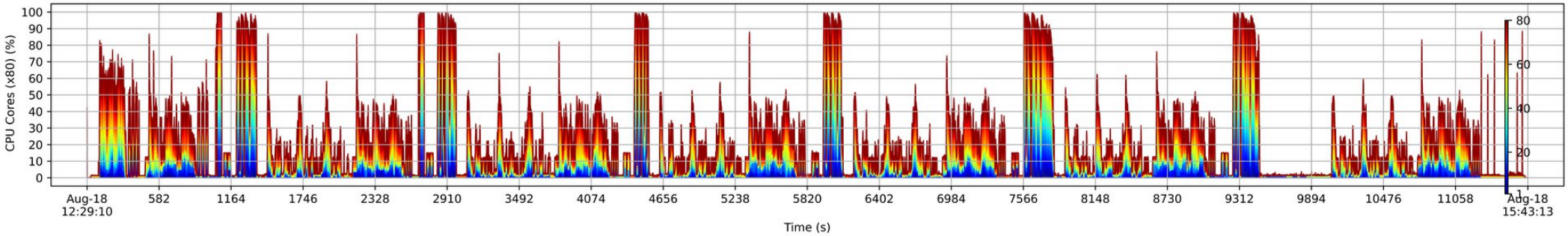
- **1 Thread:** **<program root> → main [DP3] → dp3::steps::MSWriter::WriteQueueProcess() → dp3::steps::H5ParmPredict::process().**
- **3 Threads:** **std::thread::_State_impl:ThreadPool::StartThreads → … → dp3::steps::OnePredict::PredictSourceRange()**
- **1 Thread:** **dp3::steps::MSWriter::WriteQueueProcess().**
- **8 Threads:** **dyscostman::ThreadedDyscoColumn::EncodingThreadFunctor::operator()**

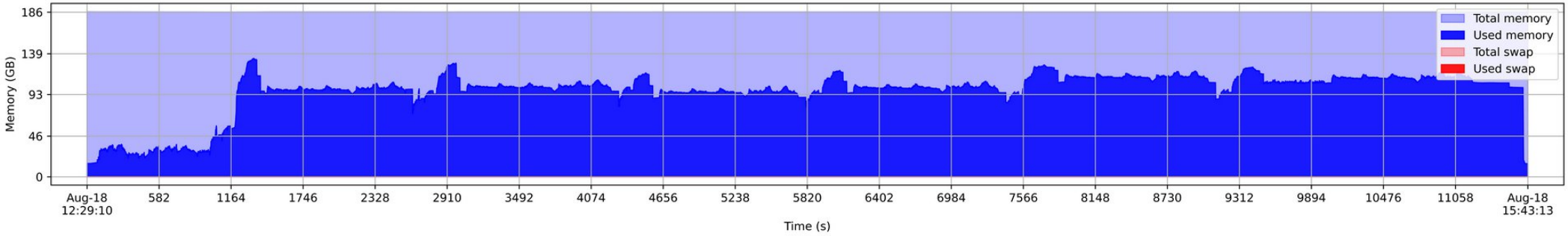


Single-node run

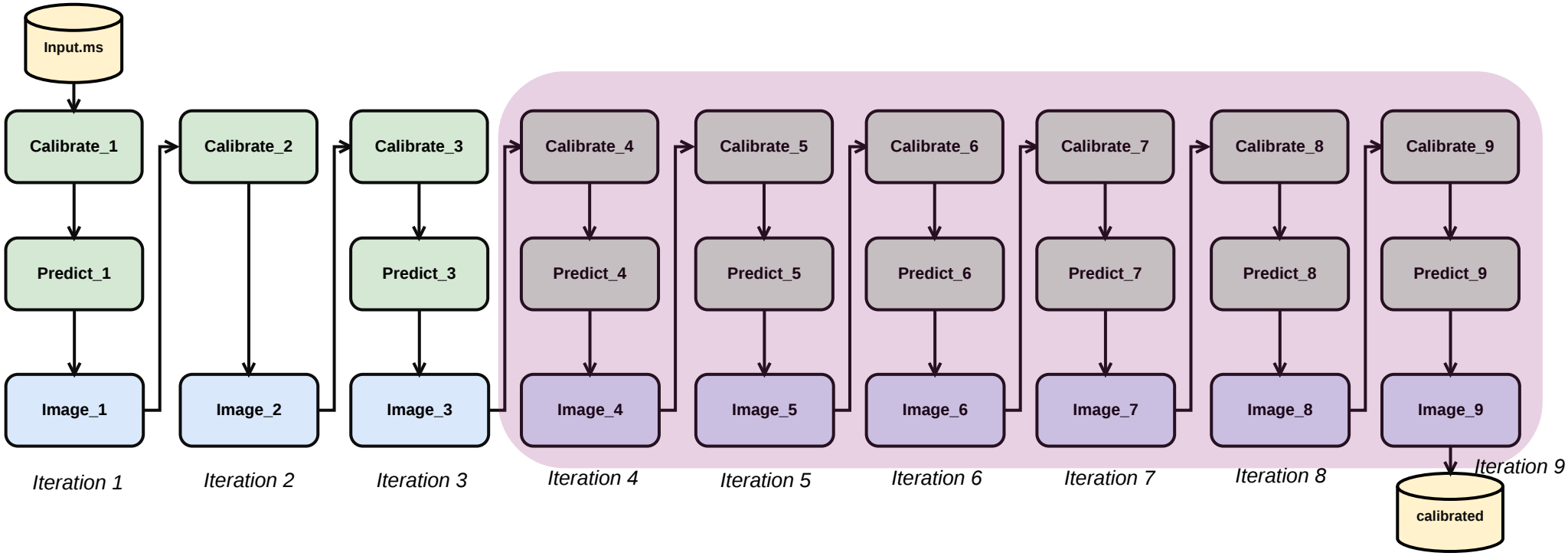
- **WSClean version 3.4.0**
- **1x Cascade Lake Node:**
  - Intel Cascade Lake 6248 CPU @ 2.50 GHz
  - Dual-Socket CPU, 20 cores per socket
  - 192 GB Memory

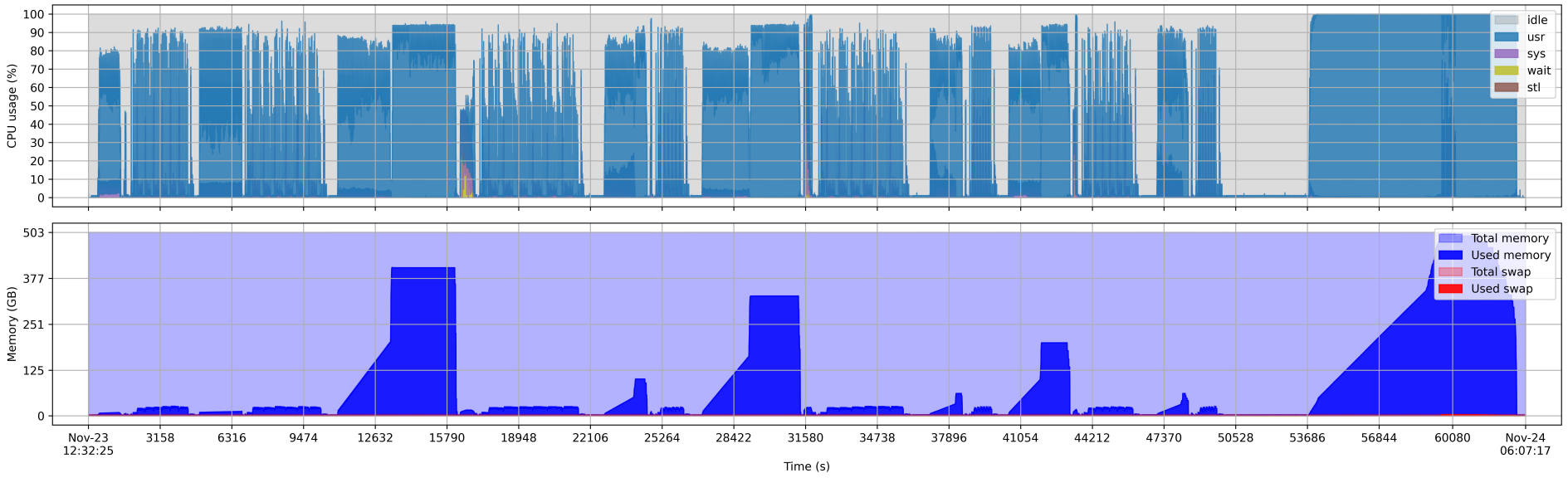






Low ICal | 9 Iterations





Conclusion

- **Benchmarking strategy** (co-design approach)
  - ✓ Automate and compare benchmarks from different systems, environments and configurations.
  - ✓ Begin with a broad approach using *benchmon*
  - ✓ Then move to more refined tools like *perf*, or *HPCToolkit*

- **SKAO pipelines:**
  - ✓ Scalability issues.
  - ✓ Additional benchmarks for the extended version of Low Ical.

THANK YOU

Anass SERHANI
anass.serhani@inria.fr

SCOOP
DP ART Team
SKA Software Engineering